J. Inst. Eng. India Ser. B (February 2022) 103(1):131–142
https://doi.org/10.1007/s40031-021-00651-0

ORIGINAL CONTRIBUTION

MapReduce-based Fuzzy *C*-means Algorithm for Distributed Document Clustering

Tanvir H. Sardar[1] · **Zahid Ansari**[2]

Received: 23 October 2020 / Accepted: 5 July 2021 / Published online: 19 July 2021
© The Institution of Engineers (India) 2021

Abstract The clustering of big data is a challenging task. The traditional clustering algorithms are inefficient for clustering big data. The recent researches in this field suggest that the traditional clustering algorithms needed to be redesigned for the modern architecture of computing. This wok has proposed a novel MapReduce-based fuzzy *C*-means algorithm for big document data clustering. The algorithm is extensively experimented with using different sizes of document datasets and executed over the Hadoop cluster of different sizes. The proposed algorithm's efficiency is compared against serial traditional fuzzy *C*-means and MapReduce-based *K*-means algorithms. The proposed design of the fuzzy *C*-means algorithm is scaled well with the Hadoop platform and documents big datasets and resulted in a performance gain.

Keywords MapReduce · Hadoop · Parallel fuzzy *C*-means · Document clustering · Distributed computing

Introduction

The process of obtaining required knowledge from the dataset by grouping data into several categories is known as the clustering process of data mining [1, 2]. This important job of data categorization based on certain objectives and the dataset is accomplished by clustering algorithms [2]. For document Big Data, the clustering job is started with the input of preprocessed documents. The documents are preprocessed in terms of unique words and their relative frequencies of presence in a particular document [3]. The clustering algorithms arrange a document dataset into groups based on a certain objective like news filtering and grouping of similar news from a digital news repository, the grouping of journal articles based on similarity of title and content, to keep only particular information while ignoring information, not in requirements, to filter out unnecessary files from the dataset, etc. [4, 5].

K-means has appeared to be one of the most widely used clustering algorithms [6]. Fuzzy *C*-means is a modification of the *K*-means algorithm using fuzzy logic. The fuzzy logic in cluster analysis states that each data point belongs to each cluster, unlike hard clustering algorithms such as *K*-means, with a varying membership value between 0 and 1. After calculating the membership value of each data points for each centroid, generally, a membership matrix is created which consists of a summarization of each data point's membership to each centroid. The new centroids are calculated based on the values of the membership matrix using the fuzzy centroid recalculation formula [7]. The execution time of fuzzy *C*-means is O (N^2) where N is the number of objects in the dataset [8].

Big Data in general is generating at a very high speed. To cluster such big datasets, the traditional clustering algorithms and architectures are proven ineffective [9–11]. It is a challenging task to make use of new architectures and paradigms and to design traditional clustering algorithms (such as fuzzy *C*-means) for obtaining better efficiency in clustering [12−14]. An effective tool for big data analysis is Hadoop. It is a distributed computing architecture that solves a problem by splitting the data and

✉ Zahid Ansari
zahid_cs@pace.edu.in

[1] School of Computer Science & Engineering, Jain University, Bangalore, India

[2] Electrical Engineering Section, University Polytechnic, Aligarh Muslim University, Aligarh, India

processing it in several computing units and accumulating results at the end [15–17]. The data split and management are provided by a Hadoop file system named HDFS and the processing part is managed by a novel programming paradigm called MapReduce [18–20]. The efficiency obtained from a Hadoop cluster is related to the intelligent design of the algorithm using MapReduce and the organization of the Hadoop cluster.

This research article provides the work of a novel MapReduce-based fuzzy *C*-means algorithm experimented over a diverse set of document datasets and Hadoop cluster sizes. One of the notable contributions of this proposed clustering algorithm is in its ability to cluster high-dimensional and large-sized document big data using fuzzy logic and MapReduce incorporated modification of fuzzy *C*-means. We have used multiple datasets for our experimentations with variable sizes ranges to 1 GB. These datasets have been experimented for the proposed algorithm on top of variable Hadoop cluster sizes ranges from 3-node cluster to 10-node. As per our knowledge, the proposed work is novel as the research corpus does not provide MapReduce-based fuzzy *C*-means algorithm experimentation as provided in this work and obtained efficiency in clustering job.

Related Works

The multiple uses of the *K*-means algorithm for many decades in clustering different datasets for different objectives of clustering are mainly due to its simplicity and ease in implementation [21]. There are many applications and datasets, including document datasets, where *K*-means is experimented with and met the required clustering objective [22–24]. The shortfall of *K*-means while clustering large datasets is the consumption of much higher execution time. This makes *K*-means inefficient for the real-world huge repository of document datasets [25]. It is also to be mentioned here that document datasets fall under big data due to its feature of high dimensionality, fully unstructured-ness, and large volume. To obtain quality clusters, a modification of the *K*-means algorithm with fuzzy logic was reported in the literature named fuzzy *K*-means [26]. *K*-means is a hard clustering algorithm that assigns each data point to a single cluster. On the other, fuzzy *K*-means is a soft clustering algorithm in which each data point belongs to each cluster with a certain degree of probability. This probability is known as membership value ranges from minimum 0 to maximum 1 [27, 28]. The recalculation of initial or last iteration generated centroid values is obtained using a formula that makes use of membership values of each object to a particular centroid. It is been a concern for traditional algorithm's scalability for big datasets. To cluster efficiently, the traditional clustering algorithms are required to be modified for new architectures and new required paradigm. This is for clustering document big data in less amount of time. The design of traditional algorithms should be correct enough to produce efficient clustering on the selected platform [29].

There are a few works reported in recent literature which presented MapReduce-based *K*-means [30–34]. The literature provides few modifications of fuzzy *C*-means for clustering document big data. Many of the modifications are merely optimizations of traditional fuzzy *C*-means. There are only a few MapReduce-based fuzzy *C*-means reported in the literature. The point is indeed too rare than *K*-means [35, 36]. In this section, we have provided the most prominent works of traditional along with MapReduce-based fuzzy *C*-means modification for document datasets.

Máximo et al. [37] have executed fuzzy *C*-means on Mahout. Mahout tool provides its implementation of fuzzy *C*-means giving authors no option in designing the fuzzy *C*-means. The experimentation is not performed enough and the authors reported that the fuzzy *C*-means is faster than *K*-means for large datasets, which is generally the opposite in theory. The more inconvenience in observation of the experiments says that the quality of clustering is poorer with fuzzy *C*-means than *K*-means. The reason for poor quality clustering is predicted by the author as due to the presence of noise in the working dataset and low quality of dimensionality reduction in the dataset. The author stated clearly that the Mahout method of dimensionality reduction is insufficient for documents and the preprocessing should be conducted using proper coding before clustering,

Zongzhen et al. [38] fuzzy *C*-means is implemented using many mappers and single reducer using MapReduce. The mapper part does the membership allocation of objects and the reducer part, after combining all mapper results, recalculates the centroids. The work designs a MapReduce-based fuzzy *C*-mean but failed in detailed experimentation. Without detailed experimentations, the effectiveness of the proposed algorithm cannot be analyzed. The authors experiment with the algorithm only with a pseudo-distributed Hadoop cluster which uses one master and only one slave instance. A total of 500 documents were used in the experimentations.

Rojcek [39] clustered document dataset using fuzzy logic that comes out from Fuzzy ART net. The membership of a data point to a cluster is measured using a specific vigilance test. This provides a different degree of belongingness. The author pointed out the large time complexity of this method. A total of 600 documents were used in the experimentations.

Mei and Wang [40] proposed a novel fuzzy *C*-means algorithm using the stochastic gradient method which clusters documents. The work is designed for online execution such that while a document arrives through the Internet immediately, the centroid is recalculated using fuzzy formula and stochastic gradient method. Two different fuzzy *C*-mean versions with different objective functions are provided. This work can produce clusters with incrementing document datasets. The work cannot be scalable as the work is dependable on traditional architecture with a standalone system. To make the work truly working for online datasets, a scalable computing framework is required with a proper design of fuzzy *C*-means for the framework. The experiments used a subset of the 20_newsgroups dataset which is small in size.

In the literature [41], Nogueira et al. proposed a method for document clustering using traditional architecture and programming paradigm. The document dataset was preprocessed to term frequency-inversed term frequency through manual coding before providing as an input to clustering. The experimentations were conducted using five datasets while each class is divided into five classes. Each class has 90 documents. The work lacks the clear distinction between its novelty and its probable capacity with real-world scalable and large document dataset clustering.

In Priandini et al. work [42], document clustering is implemented by linking fuzzy *C*-means and K-nearest neighbors method. This is to obtain higher performance in clustering. The work is executed in two steps. The first step clusters training documents using fuzzy *C*-means. The second step classifies the test documents and the output is displayed. The work is performed on small datasets and scaling for large real-world document datasets seems inadequate.

Kumari et al. [43] presented a modification to fuzzy *C*-means. A spatial function is designed for membership calculation. The spatial function value is high for a document if the majority of the neighbor documents are in the same group of this document. The spatial function is designed such that it increases the membership in the homogeneous section and the clustering output remains the same. When a document becomes misclassified, the value of the spatial function will reduce based on the impact of the neighbors of the document. Thus, giving the advantage of correctly determining the misclassified documents. The shortfall of the work is that the work is slower than traditional fuzzy *C*-means and for real-world large document datasets, there is a little probability of its effectiveness.

Karol and Mangat [44] hybridized fuzzy *C*-means and *K*-means with particle swarm optimization (PSO) and compared the result with traditional fuzzy *C*-means and *K*-means. The experiments were carried out with two standard document datasets. The performance of hybrid algorithms provided better document clusters against *K*-means and FCM without hybridization. It is stated that fuzzy *C*-means with PSO hybridization provides better quality clusters than traditional fuzzy *C*-means, *K*-means, and *K*-means with PSO.

Goyal et al. [45] contrast two techniques: cosine and fuzzy, using the *K*-means algorithm. The objective is to produce the quality clusters and to determine the best technique among these two. The quality and efficiency-wise fuzzy technique is observed better than the cosine similarity-based technique. The paper does not provide any details regarding the document dataset used for experimentations.

Win et al. [46] presented a modification of the FCM named hyperspherical fuzzy *C*-means (H-FCM) in which a novel dissimilarity measure is used. The novel dissimilarity measure is found better than the Euclidean measure in finding document proximity. The prosed algorithm produces quality clusters under different levels of fuzzy parameter values. The work is carried out on only a few documents making the results less reliable.

Based on the methodologies and clustering results obtained from these above-mentioned works, we can obtain the following observations:

- The document should be preprocessed properly before an input for clustering. It is reported that tools like Mahout sometimes provide preprocessed document datasets that are not effective in clustering [37].
- Fuzzy *C*-means, for document clustering, is parallelized using MapReduce in quite a few cases only.
- Fuzzy logic-based clustering usually takes a large time in providing clustering results on traditional architectures even with small- and moderate-sized document datasets [39–41].
- For few reported literature on document clustering using MapReduce-based fuzzy *C*-means, fewer details of experimentations, simpler algorithm design specification, only pseudo-distributed mode Hadoop installation for experimentations, and smaller document datasets for performance analysis are used [38].
- The MapReduce-based design of traditional algorithms is accomplished in each design such that the mappers compute the parallel parts of the algorithm while the reducer computes the serial parts of the algorithm.
- The demerit of serial traditional works is that these algorithms are not scalable and efficient for big data clustering. The MapReduce-based fuzzy *C*-means works, although very less reported in the literature, are not so much scalable. The design of MapReduce for the traditional algorithm is quite challenging.

The above observations derived from the literature survey drive us to the following open areas in clustering

document big data using MapReduce-based fuzzy *C*-means:

- Although there are a very small number of works reported in the literature on MapReduce-based fuzzy *C*-mean, there is no work that provided the algorithm design with figure-based representation and detailed explanation using mapper and reducer parts. In this paper, we have provided an in-depth design of the MapReduce-based fuzzy *C*-means in the methodology section. The reason for choosing a part as the mapper and another part as the reducer is also justified.
- There is no work on MapReduce-based fuzzy *C*-means which experimented on document big data with high dimensions and with the combination of standard and self-crawled datasets. In this work, one standard and five self-crawled datasets with varying sizes are used for a better understanding of results. All these datasets are preprocessed with 10,000 dimensions and then clustered using the proposed algorithm.
- No work in the literature has experimented with the proposed algorithm using a Hadoop distributed platform of variably sized clusters. In this work, the Hadoop cluster of 10 nodes is used along with 8, 5 and 3 node clusters, and a pseudo-distributed Hadoop cluster is also used. This is to determine the performance gain that happened in each stage of Hadoop cluster enlargement.

Methodology

This work provides a detailed presentation of MapReduce-based fuzzy *C*-means clustering of document big data. In the next subsections, a brief description of concepts used in these works such as MapReduce programming paradigm of Hadoop distributed platform and proposed fuzzy *C*-means needs to be discussed.

Hadoop and MapReduce Programming Paradigm

Hadoop is a distributed computing architecture that works on master–slave architecture in a network-connected cluster of nodes. The single master and multiple slave nodes are configured with storage and processing management capacity known as HDFS and MapReduce, respectively. The HDFS in master splits the input dataset into prespecified sizes, transfers to slaves, collects the output from slaves and manages the cluster using multiple configurable replications of data splits during a node failure. The data splits are processed at slave nodes to generate the final result back in master through the processing component of Hadoop named MapReduce. MapReduce executes a particular task using two parts named map and reduce. In general, the number of mappers is far more in number than the number of reducers. The mapper processes are executed on the slave nodes and mainly they are designed for processing the parallel parts of a task. The reducer's work on the intermediate outputs is gathered from mappers . The reducers are designed for serial part execution of a MapReduce task and provide the final output. The standard data format used by mappers and reducers is the key, value format. Figure 1 provides the way of MapReduce implementation on a Hadoop cluster of nodes.

The figure shows a Hadoop cluster with six nodes including one master. In this example scenario, the dataset is split by the HDFS of the master and transferred to HDFS of slave 1, slave 2 and slave 3 where mappers of MapReduce execute the parallel tasks. The result of slaves 1 and 2 are fed to slaves 4 and slave 3 results are fed to slave 5, where the MapReduce reduce part executes its assigned tasks. Each reducer creates an output file which is sent back to the master node.

Document Dataset Preprocessing

The fuzzy *C*-means work with numeric values. The document dataset is preprocessed before an input for the clustering job. The preprocessing of documents takes these steps, at a minimum:

- Tokenization: The distinctive words from all of the document files are gathered to generate a phrasebook. This process is based on the bag-of-words model.
- Weighting: This progression extricates each of the word's features in the dataset by determining each term in the word reference by a weight value. This value is obtained by the term frequency * inverse document frequency (tf*idf) technique. TF determines the recurrence of the event of a word in the specific record of the document in a dataset. Just considering tf cannot give the right outcome as the normal words like "the," "a" and so on happen all the more oftentimes in a content document yet does not mean these common words are of more significance in the document file. To get a more exact weighting of a term, tf is duplicated by idf. IDF is determined dependent on the recurrence of a term in a documented record standardized to its essence in the whole dataset. As term events are scattered exponentially, a logarithm of the weight is determined for a superior weighting value of every unique word.

Proposed MapReduce Fuzzy C-means

This work implements the *fuzzy C*-means way of clustering based on the MapReduce methodology.

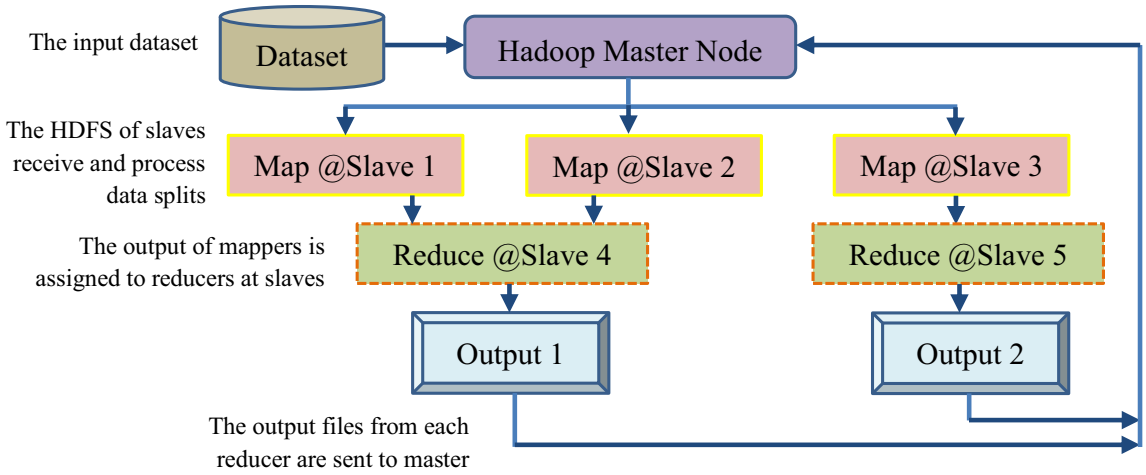
The input dataset
Dataset
Hadoop Master Node
The HDFS of slaves receive and process data splits
Map @Slave 1
Map @Slave 2
Map @Slave 3
The output of mappers is assigned to reducers at slaves
Reduce @Slave 4
Reduce @Slave 5
Output 1
Output 2
The output files from each reducer are sent to master

Fig. 1 Data processing in MapReduce on Hadoop

The serial *fuzzy C-means* stages are provided as follows:

Arbitrarily choose k data points as centroids $\odot = \{\odot_1, \odot_2, \ldots\ldots, \odot_j, \ldots\ldots, \odot_\kappa\}$, from the dataset consist of data points $O = \{O_1, O_2, \ldots, O_i \ldots, O_\eta\}$. The fuzzy coefficient is selected as m.

For each data points O_i, for each centroids $\odot_j$, compute membership matrix (MM) μ_{ij}. The number of iterations is given as the value *l*.

$$\mu_{ij} = \sum_{l=1}^{\kappa} \left(\frac{O_i - \odot_j}{O_i - \odot_l} \right)^{2/(m-1)} \tag{1}$$

Re-compute the centroids $\dot{O}_j$ for the last iterations centroid $\odot$j.

$$\dot{O}_j = \frac{\sum_{i=1}^{n} \mu_{ij}^m O_i}{\sum_{i=1}^{n} \mu_{ij}^m} \tag{2}$$

It is urgent to plan the mapper and reducer of a MapReduce task to such an extent that the parallel tasks can be allocated to mapper processed and sequential tasks to the reducer processes. On the off chance that this is cultivated appropriately, the designed algorithm will be proficient. We saw that for our proposed algorithm, the parallel task is identified as Eq. (1) (MM calculation) implementation and serial tasks are identified as Eq. (2) (re-computation of centroids). The proposed algorithm is designed based on the above observation. Figure 2 depicts the proposed algorithm.

Figure 2 displays the share of work between mappers and reducers for the proposed algorithm. Line 5 shows the process to obtain a membership matrix for each centroid and object pair. The MM value is squared in line 6 to use later in the reducer. This process will lessen the load of computing time in the reducer. The values of MM and an object for each centroid are stored in a string in line 7. Line 9 shows another string that holds all of the objects and MM values concatenated for a centroid. This step is required as the slave working on the mapper process and calculating the above-stated tasks can be used later as a reducer by the master node. The possibility also lies in sending the list one at a time using some method coded outside the mapper. As the document datasets are large enough making, we output the strings one by one to the reducer process. This is what is done in line number 13 onward. In the lines of 13 and 14 for each of the centroids objects and respective MM, values are extracted. Line number 16 is sending these strings one by one to the reducer.

The stages for reducer are an implicit gathering of sorting and shuffling followed by reducing. This sorting and shuffling phase makes the centroids number to the reducer phase. Line numbers 6 and 7 retrieve back the objects and respective MM values and start the initialization process for Eq. (2) implementation. Line numbers 8 and 9 compute the upper and lower part of Eq. (2) for each centroid. It obtains the newly calculated centroids and is kept in a list (line 12). Line 15 remits this list one at a time and stores it in a Hadoop-specific file called "part file." The input of centroids for the next iteration is obtained from this part file.

Experimental Results

To judge the effectiveness of the proposed algorithm, we have conducted many experiments using different conditions. The algorithm's comparative efficiency is evaluated based on execution time observation while changing the dataset size and Hadoop cluster size. We have set up the Hadoop cluster using commodity hardware. The commodity hardware-made nodes of the Hadoop cluster have AMD processing with the graphics card, DDR3 RAM of 8 GB size, hard disk drive capacity of 500 GB and

Algorithm for Mapper

Input:A set of document objects transformed in numeric vector form O = {O_1, O_2, ...,O_i…,O_η}, randomly chosen κ initial cluster centers Θ ={ Θ_1, Θ_2,….,…,Θ_j,……, Θ_κ} <key, value> pair input is taken where key is offset and value is the object. Each mapper also gets the initial /last iteration outputted centroid access.

Output: An intermediate <key, value> pair of (Θ_j , $\check{U}_j$) where 1 ≤j ≤ k

1. μij =0
2. <list>ǔij, <list>Ǔj =NULL
3. FOR each $\Theta_j \epsilon \Theta$ DO
4. FOR each input.value ($O_i \epsilon O$) DO
5. Calculate μ_{ij} using eq. (1)
6. $\mu^2_{ij} = \mu_{ij} * \mu_{ij}$
7. <list>$\check{u}_{ij} = O_i$ .concat(μ^2_{ij})
8. END FOR
9. <list>$\check{U}_j = \Theta_j$. concat($\check{u}_{ij}$)
10. END FOR
11. FOR each $\check{U}_j \epsilon$ <list>$\check{U}_j$ DO
12. Θ_j = extract.centroid(Ǔj)
13. Θnj=extract.centroidnumber(Θ_j)
14. <list>$\check{u}_{ij}$ =extract.MemMatrix(Ǔj)
15. FOR each $\check{u}_{ij} \epsilon$ <list>$\check{u}_{ij}$ DO
16. release(Θnj, $\check{u}_{ij}$)
17. END FOR
18. END FOR

Algorithm for Reducer

Input: <Key, Value> pair of <Θ_{nj}, <list> $\check{u}_{ij}$ > outputted by M mappers, where Θ_{nj} is Centroid number of Θj and $\check{u}_{ij}$ is the concatenated list of mapper input $O_i \epsilon$ O and its corresponding μ^2_{ij} value. This <list> $\check{u}_{ij}$ is obtained by sorting and shuffling phase of reducer on $\check{u}_{ij}$ for each Θnj.

Output: <Key, Value> pair of (NULL,Ȯ) where NULL is a space and n Ȯ are newly calculated centroid values.

1. Ɲ =0
2. ɗ=0
3. Ȯ =NULL
4. FOR each input.key (Θnj) DO
5. FOR each input.value ($\check{u}_{ij} \epsilon$ <list> $\check{u}_{ij}$) DO
6. O_i = extract.ObjectValue($\check{u}_{ij}$)
7. μ^2_{ij} = extract.MembershipValue($\check{u}_{ij}$)
8. Ɲ += $O_i * \mu^2_{ij}$
9. ɗ += μ^2_{ij}
10. END FOR
11. $\dot{O}_j$ = Ɲ / ɗ
12. Ȯ.add($\dot{O}_j$)
13. END FOR
14. FOR each $\dot{O}_j \epsilon \dot{O}$ DO
15. release(NULL, $\dot{O}_j$)
16. END FOR

Fig. 2 The MapReduce design of the proposed fuzzy C-means for document clustering

maximum 100 MB/s generating network connectivity. Each of the nodes is run on top of Ubuntu 18.04 OS. For experimentations, a set of different document datasets are used. One such dataset is a standard and widely used document dataset named 20_newsgroups, while the other five datasets are gathered by searching different net contents and copying them. All of the selected document datasets are first preprocessed using the tf*idf technique as explained in the methodology section. After preprocessing the 20_newsgroups dataset, generate an 80 MB file. This file keeps the record of 10,000 unique word's corresponding tf*idf value. Each of the other five document datasets is organized into 20 folders where each of the folders is dedicated to storing the unstructured document files of a certain domain type of online content. The content domains are primarily news and articles of books, culture, education, business, life, politics, tech, world, opinion, travel, cars, economy, etc. The documents are slowly gathered and the dataset size is increased stage by stage such that we preprocessed the datasets in phases to get the preprocessed clusterable datasets of even sizes of 100 MB, 250 MB, 500 MB, 750 MB and 1 GB. All of these document preprocessed datasets consist of uniform dimensions of 10,000. The above-spacified set of data collections have experimented against pseudo-distributed, 3-node, 5-node, 8-node and 10-node sizes of Hadoop clusters. The Hadoop pseudo-distributed node is just an image of a truly distributed Hadoop cluster in a single computing machine. A Hadoop pseudo-distributed node has a single instance of master and slave nodes running over. The experimentations on combinations of different datasets and Hadoop cluster nodes provide us differences in execution time for the proposed algorithm. This in turn enables us to evaluate the efficiency and scalability of the proposed algorithm. The number of initial centroids is selected as 20 and a total of three iterations are executed for all runs of the algorithms.

Two values of the execution time can be easily compared for evaluation using ratio calculation. We have considered the execution time of a 10-node cluster as the base value for ratio comparison concerning other Hadoop cluster's execution times. The different execution times obtained by executing different datasets for different Hadoop cluster sizes are presented in Table 1. A general

rule derived from the execution times states that the efficiency increases as fast as the Hadoop cluster size is enlarged. For instance, a 10-node Hadoop cluster able to complete the data clustering task 3.1 times quicker as compared to a pseudo-distributed computing cluster besides nearly 2.5 times quicker as compared to a 3-node computing cluster. The above observation was made when a 1 GB document dataset was used for the clustering task. This investigative clarification of the outcome for the proposed algorithm lies in the mapper and reducer phase's design. The proposed algorithm's mapper is designed in such a way that the membership matrix construction-related computations are performed there in Eq. (1) and the re-computation of centroids in the reducer as in Eq. (2) after obtaining output from the mapper. The reducer phase as it implements Eq. (2) consumes more and more time than the mapper. As stated earlier, the dataset is having a large 10,000 dimensions. This high-dimensionless dataset, other than its size, is another reason that the proposed algorithm takes much time before completion. The distance calculation takes much time because the calculation of distance in mappers and reducer takes a lot of time for dimension to dimension-based calculations between the centroids and the data objects. The time consumption would be much lesser if the number of dimensions was less even if the same dataset size was clustered.

The data of Table 1 are presented from a different point of view in Table 2. Table 2 displays the number of times that a higher dataset takes to complete the clustering task for each category of execution style. This is displayed using ratio calculations. As an example, it can be observed that the 1 GB dataset consumes 21.42 times more execution time than while clustering the 20_newsgroups dataset using a 10-node Hadoop cluster.

To examine the efficiency of the projected algorithm, data scale-up is portrayed employing a line graph in Fig. 3. Data scale-up delivers the reader through a perfect depiction of improvement in efficiency for different sized datasets and Hadoop execution style. This permits to properly examine the proposed algorithm's efficiencies. The X-axis of Fig. 3 signifies the number of diverse sizes of datasets considered for experimentations, and the Y-axis of Fig. 3 signifies the quantity of the execution time recorded from diverse types of Hadoop cluster executions of proposed algorithms. The lines in Fig. 3 are colored and these colors are used for representing a particular type of Hadoop cluster executions of the proposed algorithm. Figure 3 shows the clustering job is completed in a certain time and its execution time is proportional to the dataset size of its clusters. This remark delivers the notion that the proposed clustering algorithm is scalable for diversely sized and high-dimensional document big data.

One can get the clear understanding from the tables and figures provided above that the proposed clustering algorithm design is successful for large and high-dimensional document datasets. It is also observable that the Hadoop cluster is efficient both for standard and big datasets. However, it is also observed from the line graph that the scale-up is not linear. Nonlinearity in the sense that for the same amount of enlargement of dataset size, the similar growth of time complexity is not observed. For an instance, while executing the 500 MB dataset using proposed algorithm on top of 10-node cluster, we see that 2130 s on time is consumed. For the same setup of execution style with 1 GB dataset, it took 3492 s. So the dataset size is doubled

Table 1 Execution time and performance ratio concerning different execution strategy

Dataset size and ratio	MapReduce fuzzy C-means execution time in second				
	Pseudo-distributed	3-node cluster	5-node cluster	8-node cluster	10-node cluster
20_newsgroups	424	357	300	222	163
Ratio	2.6	2.19	1.84	1.36	1
100 MB	638	560	439	314	224
Ratio	2.85	2.55	1.96	1.4	1
250 MB	2224	1856	1520	1142	794
Ratio	2.8	2.33	1.91	1.43	1
500 MB	6219	5204	4124	3052	2130
Ratio	2.92	2.44	1.93	1.43	1
750 MB	8580	7262	5935	4158	2869
Ratio	2.99	2.53	2.06	1.45	1
1 GB	10,827	9009	6879	4958	3492
Ratio	3.1	2.58	1.97	1.42	1

Table 2 Alternative view: performance ratio with respect to different execution strategy

Node(s)	Dataset					
	20_newsgroups	100 MB	250 MB	500 MB	750 MB	1 GB
Pseudo-distributed	424	638	2224	6219	8580	10,827
Ratio	1	1.5	5.2	14.66	20.23	25.53
3-node cluster	357	560	1856	5204	7262	9009
Ratio	1	1.56	5.19	14.58	20.34	25.23
5-node cluster	300	439	1520	4124	5935	6879
Ratio	1	1.46	5.0	13.75	19.78	22.93
8-node cluster	222	314	1142	3052	4158	4958
Ratio	1	1.41	5.14	13.75	18.73	22.33
10-node cluster	163	224	794	2130	2869	3492
Ratio	1	1.37	4.87	13.0	17.6	21.42

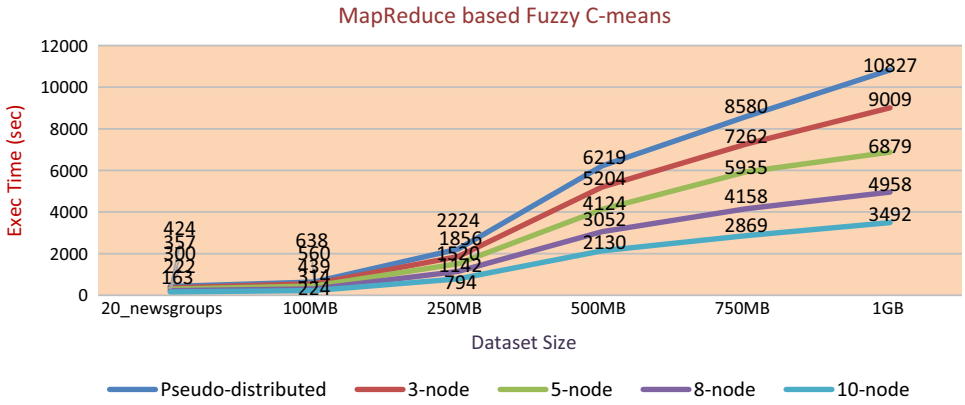

Fig. 3 Line graph of data scale-up

but the execution time requirement was not doubled but much lesser of 1.64 times. For experiments with diverse datasets and Hadoop cluster setups, the same observation on nonlinearity of the execution is recorded. The main underlying reason of this nonlinearity is the high time consumption of proposed algorithm's reducer and the process and network overhead of a Hadoop cluster nodes.

The novelty of the work can be judged by the present works of this domain of literature for document clustering using MapReduce-based fuzzy *C*-means. None of the present works has described a MapReduce-based fuzzy *C*-means design in detail and experimented with multiple datasets and different sized Hadoop clusters. This also makes it challenging to evaluate the efficiency of proposed MapReduce-based fuzzy *C*-means with other works of document clustering. We have designed two techniques for evaluation of the efficiency of the proposed work: (a) the proposed MapReduce version of fuzzy *C*-means is experimentally compared and contrasted against its non-MapReduce-based serial traditional version. And, (b) the proposed MapReduce version of fuzzy *C*-means is experimentally compared against a recent work [47] of MapReduce-based non-fuzzy version of the same algorithm. The above-specified comparisons of the proposed MapReduce version of fuzzy *C*-means are explained in detail in the following two subsections:

*Comparison of proposed MapReduce Version of Fuzzy C-*Means *against its Non-MapReduce-based Serial Traditional Version*

The proposed MapReduce version of fuzzy *C*-means is experimentally evaluated concerning its non-MapReduce-based serial traditional version to judge the efficiency differences. In this process, the variations in the number of nodes and the execution time gained from the Hadoop cluster computing are maintained the same for the proposed algorithm which is compared concerning standalone serial execution of the same algorithm. Similarly, the same document datasets are also used while evaluating both algorithms. The execution time for both algorithms is presented in Table 3. The second column of Table 3 is the

data of execution time obtained by executing the traditional serial fuzzy *C*-means algorithm.

The observations of Table 3 give us insight that the MapReduce version of the algorithm is faster than the traditional version of the algorithm when the MapReduce-based algorithm is executed over a 3-node Hadoop cluster and more. However, the traditional version of the algorithm is faster than the pseudo-distributed Hadoop processing of the algorithms. The cause for this is the processor, RAM and network communication overhead of one standalone system's master and slave daemon processes execution which takes place in a pseudo-distributed node. Developing the MapReduce-based algorithms and their testing requirements can only be attained using this pseudo-distributed Hadoop execution mode.

Another observation that is prevalent from Table 3 is that the larger the Hadoop cluster and the big the dataset, the better the performance of the MapReduce-based algorithms. It can be seen that the 10-node Hadoop cluster execution of the proposed MapReduce-based algorithm is 2.55 times faster than the traditional version of the algorithms for the 20_newsgroups dataset (which is just 80 MB). The same 10-node Hadoop cluster execution of the proposed MapReduce-based algorithm is 3.07 times faster when the dataset size is increased to 1 GB. It can be seen from Table 3 that this efficiency gain is linearly increasing for 100 MB, 250 MB, 500 MB and 750 MB also. This observation justifies that the proposed algorithm is efficient and more scalable than its traditional version.

Comparison of the Proposed MapReduce version of Fuzzy C-Means against a recent Work of MapReduce-based K-Means Algorithm

In a recently published work of 2020 [47], a MapReduce-based *K*-means is presented with a similar dataset and the same Hadoop cluster nodes as part of its execution strategy. This work of MapReduce-based *K*-means reported the following execution time details as provided in Table 4.

Table 4 data of execution time for MapReduce-based *K*-means can be easily compared against Table 1 data of MapReduce-based fuzzy *C*-means. It can be seen that the MapReduce version of fuzzy *C*-means is quite slower than MapReduce-based *K*-means. The execution time for each case of Hadoop clusters and dataset sizes shows that MapReduce-based *K*-means perform the clustering job at a very faster speed as compared to MapReduce-based fuzzy *C*-means. Let us take an example for 100 MB dataset execution in pseudo-distributed Hadoop cluster execution. It took 638 s to perform the clustering job by MapReduce-based fuzzy *C*-means which is accomplished by MapReduce-based *K*-means in just 151 s. This gap between the execution time among these two algorithms is increasing as the dataset size is enlarging. The reason for this gap between the execution time can be summarized between the below mentioned two key points:

- The differences between the Hadoop cluster's commodity hardware: Although the RAM and network connectivity part are the same for the nodes of the Hadoop cluster used to experiment with both of the algorithms, the processor capacity is different. The commodity hardware of MapReduce-based fuzzy *C*-

Table 3 Execution time and performance ratio for different execution strategy

Dataset size and ratio	Traditional fuzzy C-**means**	MapReduce fuzzy C-means execution time in second				
		Pseudo-distributed	3-node cluster	5-node cluster	8-node cluster	10-node cluster
20_newsgroups	417	424	357	300	222	163
Ratio	2.55	2.6	2.19	1.84	1.36	1
100 MB	623	638	560	439	314	224
Ratio	2.78	2.85	2.55	1.96	1.4	1
250 MB	2210	2224	1856	1520	1142	794
Ratio	2.78	2.8	2.33	1.91	1.43	1
500 MB	6117	6219	5204	4124	3052	2130
Ratio	2.87	2.92	2.44	1.93	1.43	1
750 MB	8491	8580	7262	5935	4158	2869
Ratio	2.96	2.99	2.53	2.06	1.45	1
1 GB	10,754	10,827	9009	6879	4958	3492
Ratio	3.07	3.1	2.58	1.97	1.42	1

Table 4 Performance of MapReduce-based K-means algorithm as in [32]

Dataset size and ratio	MapReduce-based K-means execution time in second				
	Pseudo-distributed	3-node cluster	5-node cluster	8-node cluster	10-node cluster
100 MB	151	140	120	79	52
Ratio	2.9	2.7	2.3	1.51	1
250 MB	195	164	140	114	87
Ratio	2.24	1.88	1.6	2.19	1
500 MB	520	300	176	135	110
Ratio	4.7	2.72	1.6	1.22	1
750 MB	637	324	274	165	133
Ratio	4.8	2.43	2	1.24	1
1 GB	649	403	365	180	141
Ratio	4.6	2.85	2.58	1.27	1

means algorithm's Hadoop cluster is made up of the AMD Ryzen 3pro processor which is lower in speed than the Intel Core 2 Duo processors which are used for the experimentations of the MapReduce-based *K*-means algorithm's Hadoop cluster.

- The design differences between these two algorithms: The MapReduce-based *K*-means design (as given in methodology of [47]) for mapper and reducer is quite simple in comparison with our proposed MapReduce-based fuzzy *C*-means algorithm. The mapper of MapReduce-based *K*-means calculates distances between data objects and centroids and fetches the data to the reducer. The reducer of MapReduce-based *K*-means just adds up partially obtained centroid data and divides by the number of objects added in it to find out new centroid values. On the other hand, the mapper of MapReduce-based fuzzy *C*-means executes quite higher complex manipulation of membership matrix for each centroid and data object pair. The mappers yield huge series of membership matrix assessment values to the reducers. This stage takes bulky network time. The reducer of MapReduce-based fuzzy *C*-means also performs a quite higher complex work of centroid recalculation using Eq. (2). As provided in Eq. (2), the reducer needs to calculate the multiplication, summation, and proximity calculation part which is, in comparison with the MapReduce-based *K*-means, highly complex. However, the cluster quality of fuzzy-based soft clustering of MapReduce-based fuzzy *C*-means is higher than the hard cluster quality output of the MapReduce-based *K*-means, even in presence of noise in data.

Conclusion and Future Work

The age of big data requires the novel design of traditional clustering algorithms on new architectures to obtain efficiency in clustering time. The MapReduce-based modification of clustering algorithms is recently developed and experimented with over the Hadoop platform. We have designed and experimented with the MapReduce-based fuzzy *C*-means for document big data. The wide-ranging experiments were conducted on five different types of datasets and five different-sized Hadoop clusters. The experimental evaluation shows that the proposed algorithm is scalable with Hadoop and obtained clustering jobs efficiently. The analysis of experimental data shows that the performance gain is uneven concerning cluster size and dataset size due to Hadoop cluster overhead and algorithmic design. It is also found that to obtain performance gains, large data size and bigger Hadoop clusters are more effective. The main contribution of this work lies in the experimentation of the proposed algorithm on big data of document type for clustering. There is no such work reported in the literature with such detailed design and details of MapReduce-based fuzzy *C*-means algorithm and such large-scale experimentations for many datasets, each with 10,000 dimensions and different Hadoop clusters.

As future work, the proposed algorithm can be extended with MapReduce-based fuzzy C-medoids. The design of fuzzy C-medoids can extensively experiment for performance gain with the same dataset size and Hadoop cluster size. Further, cluster quality analysis of fuzzy *C*-means and fuzzy C-medoids can be obtained and compared using different fuzzy metrics of cluster validation such as Xie-Beni and Fukuyama-Sugeno. The work can be compared with big data tools such as Spark and Mahout for the same algorithm and dataset.

Funding Not Applicable.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. L. Rutkowski, M. Jaworski, P. Duda, *Stream Data Mining: Algorithms and Their Probabilistic Properties* (Springer, Cham, 2020)
2. T.H. Sardar, A.R. Faizabadi, Z. Ansari, An evaluation of mapreduce framework in cluster analysis, in *2017 IEEE International Conference on Intelligent Computing. Instrumentation and Control Technologies (ICICICT), Kannur, India* (2017)
3. L.M.Q. Abualigah, *Feature Selection and Enhanced Krill Herd Algorithm for Text Document Clustering* (Springer, Berlin, 2019)
4. S. Bisht, A. Paul, Document clustering: a review. Int. J. Comput Appl. **2013**, 73–11 (2013)
5. M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques. KDD Workshop on text mining, 400–401 (2000).
6. J. Zhang et al., A parallel clustering algorithm with mpi-mk-means. J. Comput. **10**, 8–1 (2013)
7. A. Zahid, A. Rimaz Faizabadi, A. Afzal, Fuzzy c-Least Medians clustering for discovery of web access patterns from web user sessions data. Intell. Data Anal. **21**(3), 553–575 (2017)
8. A.S. Shirkhorshidi, et al., Big data clustering: a review, in *International conference on computational science and its applications*. Springer, Cham, 2014.
9. T.H. Sardar, Z. Ansari, Partition based clustering of large datasets using MapReduce framework: an analysis of recent themes and directions. Fut. Comput. Inf. J. (2018).
10. A. Holmes, *Hadoop in Practice*. Manning Publications Co., 2012.
11. N.M.F. Fureshi et al., An aggregate mapreduce data block placement strategy for wireless IoT edge nodes in smart grid. Wirel. Person. Commun. **106**(4), 2225–2236 (2019)
12. T. H. Sardar, Z. Ansari, A. Khatun, An evaluation of Hadoop cluster efficiency in document clustering using parallel K-means, in IEEE 2014 International Conference on IMpact of E-Technology on US (IMPETUS).
13. M. H. Singh, Clustering of text documents by implementation of K-means algorithms. Streamed Info-Ocean 1(1) (2016).
14. R. Balabantaray, R. Chandra, C. Sarma, M. Jha. Document clustering using K-means and K-medoids arXiv preprint: arXiv:1502.07938 (2015).
15. T. H. Sardar, Z. Ansari, Detection and confirmation of web robot requests for cleaning the voluminous web log data, in *2014 International Conference on the IMpact of E-Technology on US (IMPETUS)*.
16. T.H. Sardar, Z. Ansari, An analysis of MapReduce efficiency in document clustering using parallel K-means algorithm. Fut. Comput. Inf. J. **3**(2), 200–209 (2018)
17. W. Wiharto, E. Suryani, The comparison of clustering algorithms K-means and Fuzzy C-means for Segmentation Retinal Blood Vessels. Acta Informatica Medica **28**(1), 42 (2020)
18. G. Ball, D. Hall, A clustering technique for summarizing multivariate data. Behav. Sci. **153**, 12 (1967)
19. Surve and Paddune, A survey on hadoop assisted K-means clustering of hefty volume images. Int. J. Comput. Sci. Eng. (IJCSE). **113**, 6–3 (2014)
20. Li et al., K-means clustering with Bagging and MapReduce, in 44th Hawaii International Conference on System Sciences (HICSS). 1–8. 2011.
21. J.E. Judith, J. Jayakumari, An efficient hybrid distributed document clustering algorithm. Sci. Res. Essays **10**(1), 14–22 (2015)
22. Zhou et al., Large-scale data sets clustering based on MapReduce and hadoop. J. Comput. Inf. Syst. **5956**, 7–16 (2011)
23. P. Anchalia, Improved MapReduce K-means Clustering Algorithm with Combiner, in 16th International Conference on Computer Modeling and Simulation (UKSim), pp. 386–391 (2014).
24. R. M. Esteves, C. Rong, Using Mahout for clustering Wikipedia's latest articles, in *3rd IEEE International Conference on Coud Computing Technology and Science* (2011).
25. H. Zongzhen, Z. Weina, D. X. Liyue, Yanfan, A fuzzy approach to clustering of text documents based on MapReduce, in *International Conference on Computational and Information Sciences* (2013).
26. M. Rojcek, System For Fuzzy Document Clustering And Fast Fuzzy Classification, in *15th IEEE International Symposium on Computational Intelligence and Informatics*, 19–21 November, 2014, Budapest, Hungary (2014).
27. J.-P. Mei, Y. Wang, Hyperspherical fuzzy clustering for online document categorization, in 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE). IEEE (2016).
28. T. M. Nogueira, S. O. Rezende, H. A. Camargo, On the use of fuzzy rules to text document classification, in *2010 10th International Conference on Hybrid Intelligent Systems*. IEEE (2010).
29. N. Priandini, B. Zaman, E. Purwanti, Categorizing document by fuzzy C-means and Knearest neighbors approach. AIP Conf. Proc. **1867**(1), 1 (2017)
30. K. Sathiya Kumari, V. Preamsudha, G. Manimekalai, Unsupervised approach for document clustering using modified fuzzy C mean algorithm. Int. J. Comput. Org. Trends **11**, 1 (2011)
31. S. Karol, V. Mangat, Evaluation of text document clustering approach based on particle swarm optimization. Open Comput. Sci. **3**(2), 69–90 (2013)
32. M. M. Goyal, et al., *Comparison clustering using cosine and fuzzy set based similarity measures of text documents*. arXiv preprint arXiv:1505.00168 (2015).
33. T. T. Win, L. Mon, Document clustering by fuzzy c-mean algorithm, in *2010 2nd International Conference on Advanced Computer Control, Vol. 1*. IEEE (2010).
34. Pu. Zhang, Q. Shen, Fuzzy c-means based coincidental link filtering in support of inferring social networks from spatiotemporal data streams. Soft. Comput. **22**(21), 7015–7025 (2018)
35. P. Ghuli, A. Shukla, R. Kiran, S. Jason, R. Shettar, Multidimensional canopy clustering on iterative MapReduce framework using Elefig tool. IETE J. Res. **61**(1), 14–21 (2015)
36. S. Panwar, N. Nain, A novel segmentation methodology for cursive handwritten documents. IETE J. Res. **60**(6), 432–439 (2014)
37. A. Sikander, R. Goyal, M. Mehrotra, G. Parmar, Discrete-time system approximation using hybrid method based on fuzzy C-means clustering. J. Inst. Eng. Ser. B **1**, 1–9 (2021)
38. Q. Mahmood, A. Chodorowski, M. Persson, Automated MRI brain tissue segmentation based on mean shift and fuzzy c-means using a priori tissue probability maps. IRBM **36**(3), 185–196 (2015)
39. A.L.M. Pavan, A. Vacavant, A.P. Trindade, D.R. de Pina, Fibroglandular tissue quantification in mammography by optimized fuzzy C-means with variable compactness. Irbm **38**(4), 228–233 (2017)
40. F. Kayaalp, P. Erdogmus, Benchmarking the clustering performances of evolutionary algorithms: a case study on varying data size. IRBM **41**(5), 267–275 (2020)

41. B. Biswal, P.K. Dash, B.K. Panigrahi, Time frequency analysis and non-stationary signal classification using PSO based fuzzy C-means algorithm. IETE J. Res. **53**(5), 441–450 (2007)
42. S. Sampathkumar, R. Rajeswari, An automated crop and plant disease identification scheme using cognitive fuzzy C-means algorithm. IETE J. Res. **1**, 1–12 (2020)
43. A. Rajendran, N. Balakrishnan, M. Varatharaj, Malleable fuzzy local median C means algorithm for effective biomedical image segmentation. Sens. Imag. **17**(1), 1–14 (2016)
44. S. Raj, D.S. Vinod, B.S. Mahanand, N. Murthy, Intuitionistic fuzzy C means clustering for lung segmentation in diffuse lung diseases. Sens. Imag. **21**(1), 1–16 (2020)
45. S. Dong, D. Zhou, W. Ding, J. Gong, Flow cluster algorithm based on improved K-means method. IETE J. Res. **59**(4), 326–333 (2013)
46. V. Kumar, J.K. Chhabra, D. Kumar, Automatic unsupervised feature selection using gravitational search algorithm. IETE J. Res. **61**(1), 22–31 (2015)
47. T.H. Sardar, Z. Ansari, An analysis of distributed document clustering using MapReduce based K-means algorithm. J. Inst. Eng. Ser. B **101**(6), 641–650 (2020)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.